
Microsoft®
Research
Faculty
Summit
2003

Righting Software:
Tools to Improve Software Development

James Larus
SPT & PPRC
Microsoft Research
July 2003

Microsoft Research Faculty Summit 2003

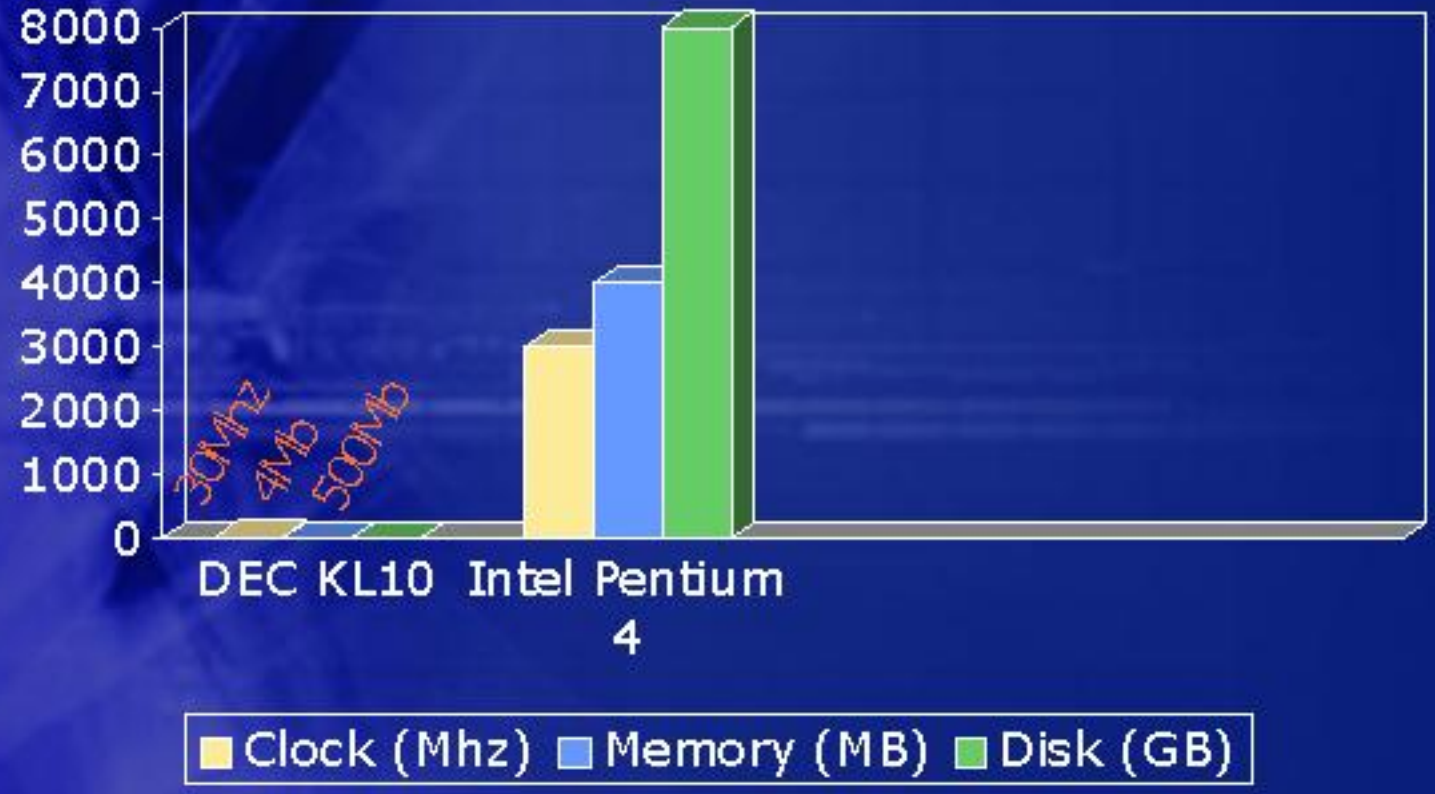
Software Development Tools Today



level
er

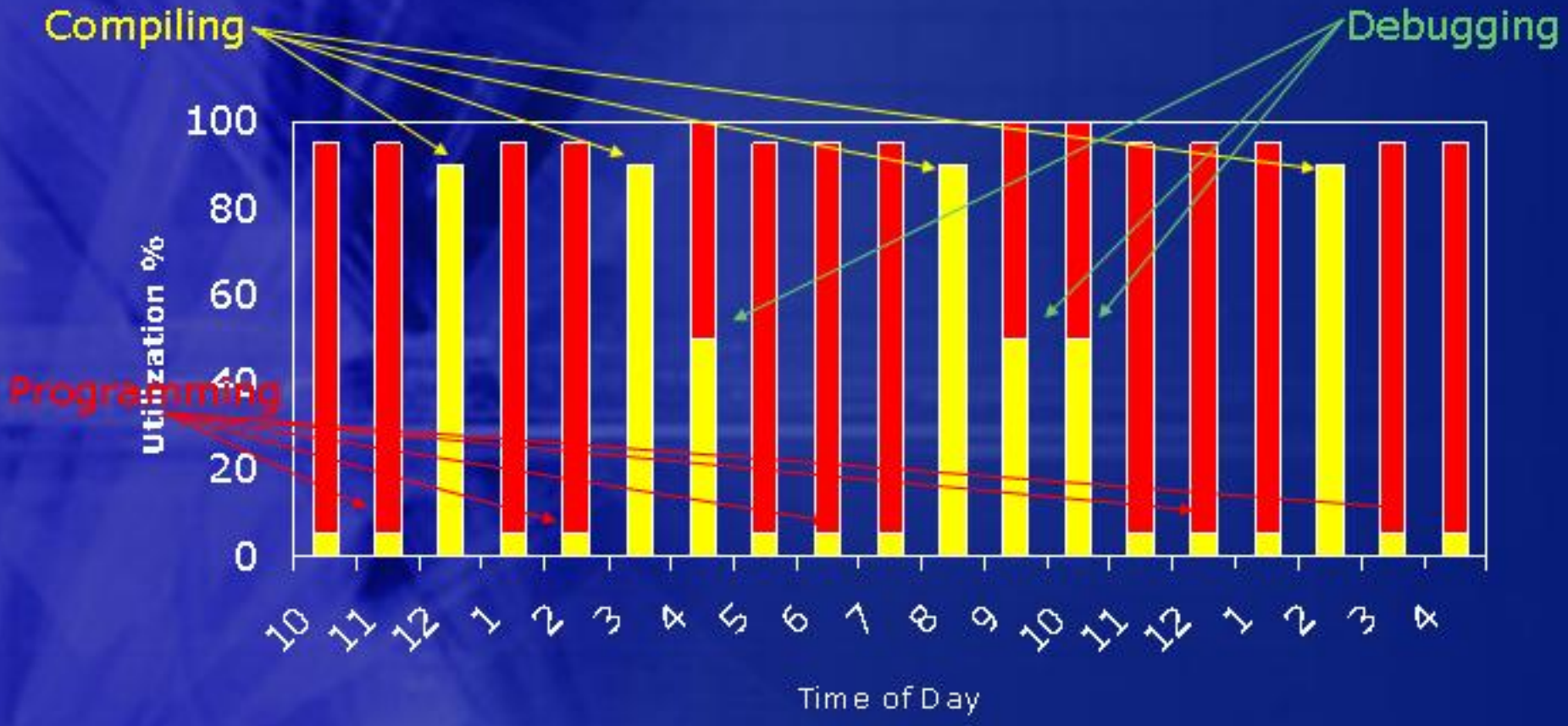
Research Faculty Summit 2003

Processor Performance 1978-2003



Microsoft Research Faculty Summit 2003

Under-Utilized Computers
Over-Utilized Developers



Microsoft Research Faculty Summit 2003

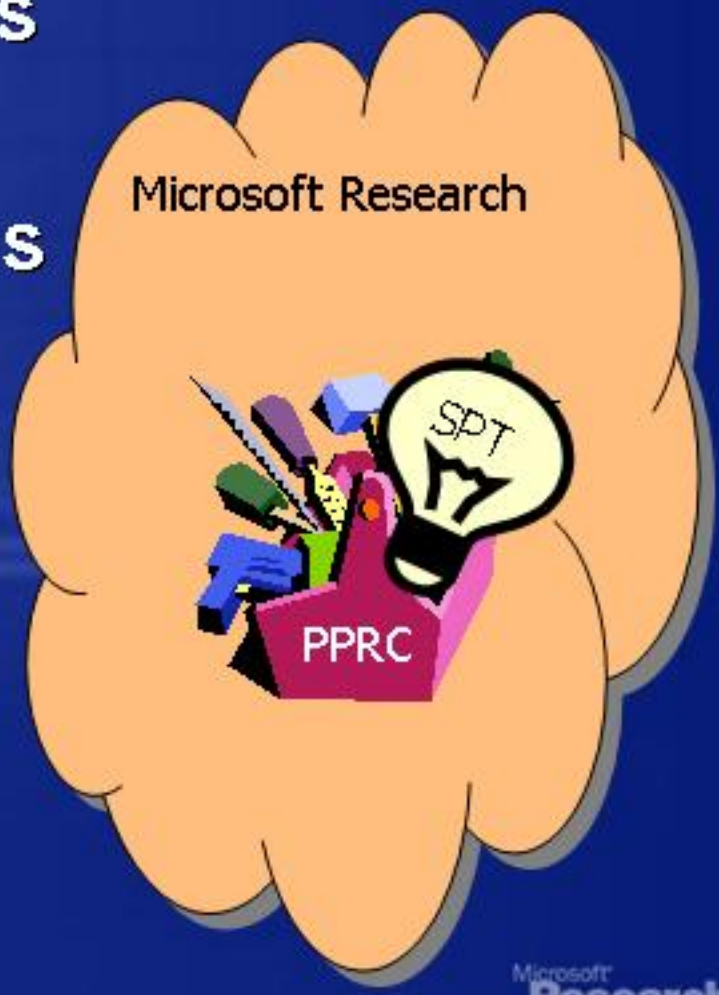
Tools for Software Development

- Apply computing power to software development
  - aid, not replace, developers & testers
- Amplify human effort
  - manage details
  - find inconsistencies
  - ensure quality

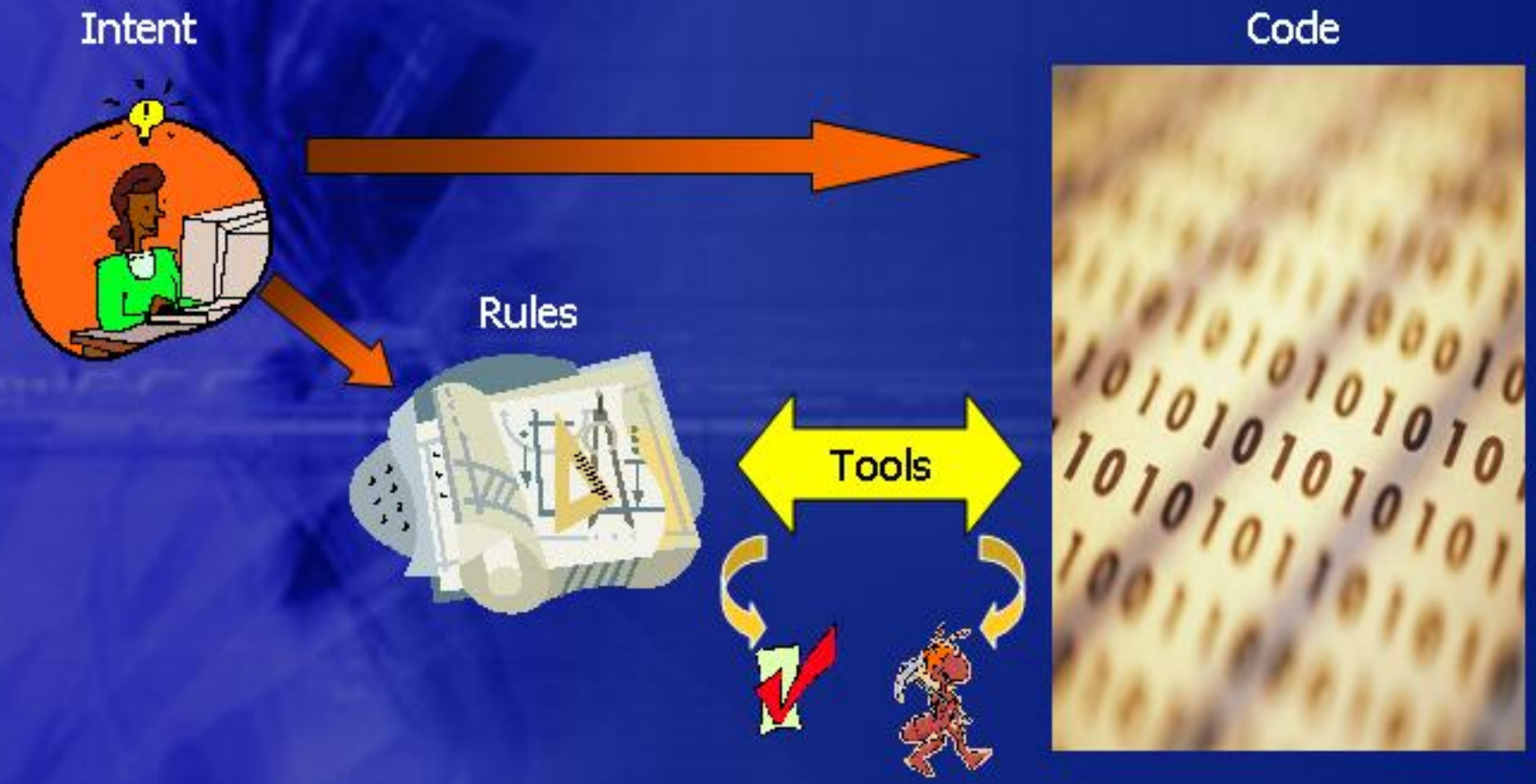
Microsoft Research Faculty Summit 2003

3 Generations of PPRC Correctness Tools

- Scalable, heuristic tools
  - PREfix, PREfast
- Sound, declarative tools
  - SLAM, Fugue, ESP
- Targeted tools
  - concurrency, security



Tools Bridge Intent and Code



Microsoft Research Faculty Summit 2003

Types of Errors

- Language usage
  - uninitialized variable, null dereferences, …
- API usage
  - close file twice, ignore error result, hold lock, …
- Semantic errors
  - deadlock
- …
- Incorrect computation (≈program verification)
  - NOT our goal!

Microsoft Research Faculty Summit 2003

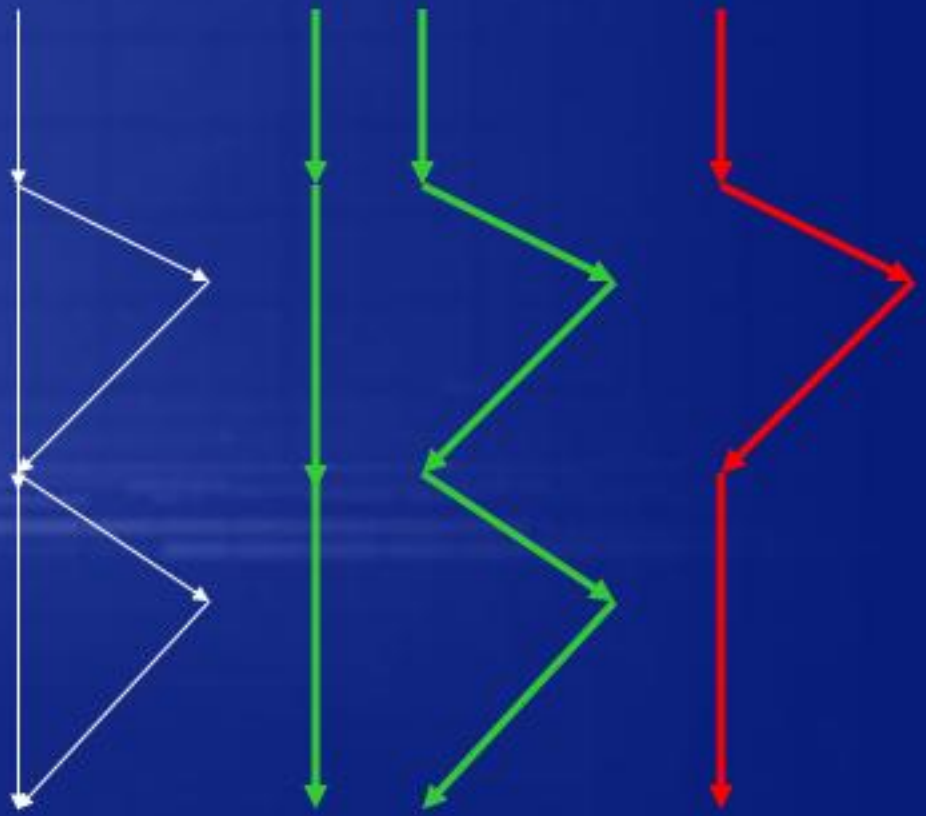
Static Program Analysis

- Tool explores all possible executions of program
  - complements run-time checking
    - completeness vs. precision
  - analyses exist for only a few properties
- Choose at most one:
  - sound analysis finds all instances of error
  - complete analysis reports no false errors
  - can't have both: Turing halting problem

Microsoft Research Faculty Summit 2003

Static Analysis Limitations

```
FILE* f;
if (complex_calc1())
    f = fopen(...);
...
if (complex_calc2())
    fclose(f);
```



Microsoft Research Faculty Summit 2003

Outline

- 1st generation tools (scalable, heuristic)
  - Prefix
  - Prefast
- 2nd generation tools
- 3rd generation tools
- Conclusion

Microsoft Research Faculty Summit 2003

PPRC's First Correctness Tools

- PREfix
  - detailed, path-by-path interprocedural analysis
  - heuristic (unsound, incomplete)
  - expensive (4 days on Windows)
  - effective
- PREfast
  - user-supplied plug-ins traverse AST
  - desktop use, easily customized
- Widely deployed in Microsoft
  - 1/6 of bugs fixed in Windows Server 2003 found by these tools

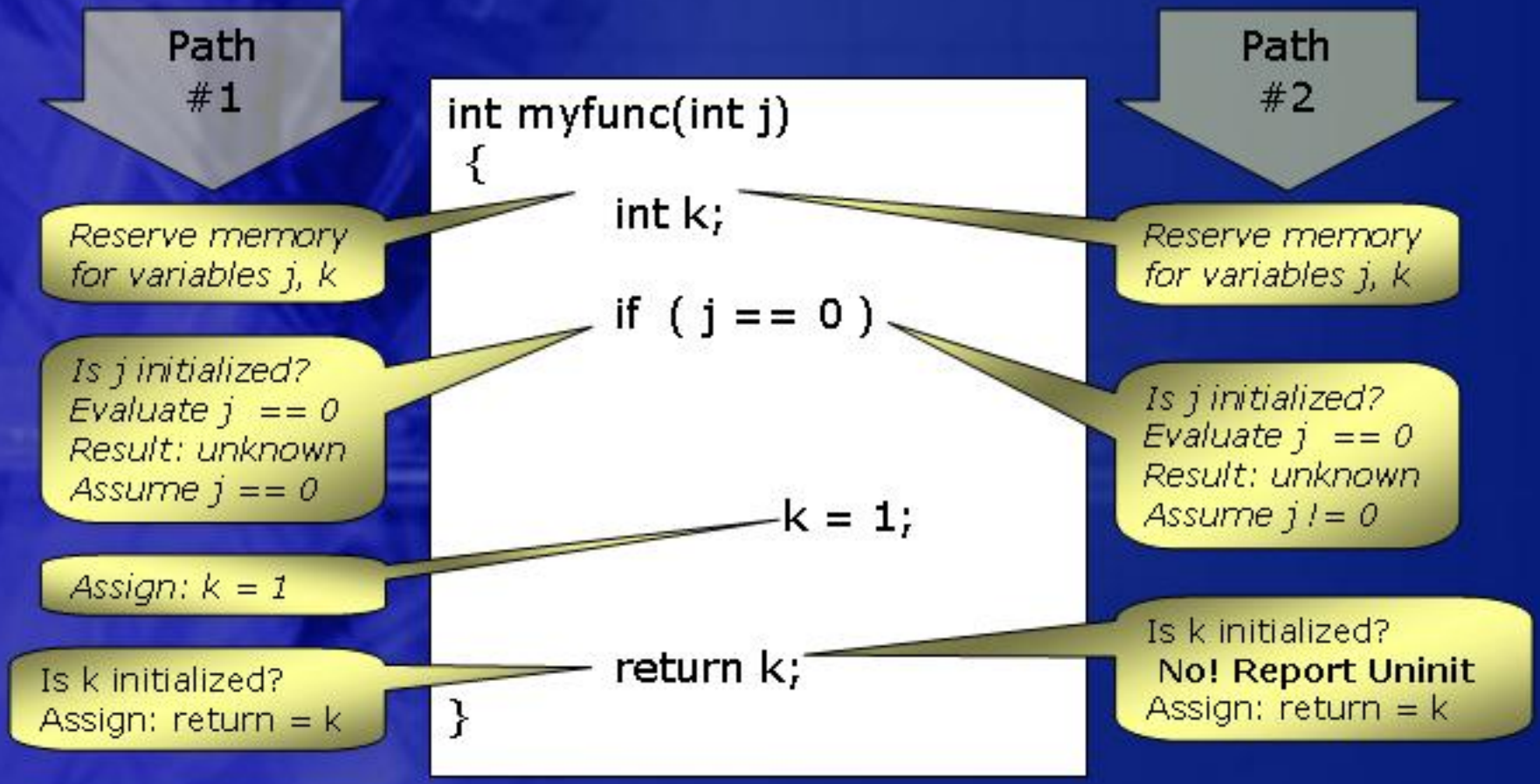
Microsoft Research Faculty Summit 2003

PREfix

- Detects errors in C/C++ code
  - null pointer, memory allocation, uninitialized value, resource state, library usage, …
- Interprocedural
  - bottom up traversal of call graph
  - models routine by examining limited set of paths (100)
  - uses model at call site
  - expensive, batch computation for large systems
- Large effort to minimize effect of false positives
  - filtering and prioritizing error reports
  - heuristics tuned to reduce noise (at cost of precision)

Microsoft Research Faculty Summit 2003

PREfix Example



Microsoft Research Faculty Summit 2003

PREfast

- Framework for local analysis of C/C++ code
  - AST from Visual Studio C/C++ compiler
  - plug-ins traverse trees, looking for idioms
  - basis for building sophisticated tools
    - familiar
    - integrated in build processes
- Low cost checking
  - local (function by function)
  - simple heuristics
- Easy extensibility
  - large community

Microsoft Research Faculty Summit 2003

PREfast Example

What's wrong with this code?

```
char *p1 = new char[10];
char *p2 = new char[10];
...
delete[] p2;
delete[] p1;
```

Can leak memory *<pointer>* due to an exception from second new operation.

```
char *p1 = new char[10];
char *p2;
try { p2 = new char[10]; }
catch (std::bad_alloc *e) {
 delete[] p1;
}
...
delete[] p2;
delete[] p1;
```

Yes, its ugly,
and necessary in a non-GC language.

Microsoft Research Faculty Summit 2003

SpecStrings

- Company-wide effort to annotate C/C++ APIs
  - buffer overruns
  - built on PREfast infrastructure
  - standard annotation language (SAL)

```
BOOL WINAPI SetupGetStringFieldW(
        ...
        __ecount(ReturnBufferSize)
        OUT PWSTR ReturnBuffer,
        IN DWORD ReturnBufferSize,
        ...);

WCHAR szPersonalFlag[20];
...
SetupGetStringFieldW(&Context,1,szPersonalFlag,50,NULL);
```

NT# 58762 PREfast: warning 202: Buffer overrun for stack buffer 'szPersonalFlag' in call to 'SetupGetStringFieldW': length 100 exceeds buffer size 40.

Microsoft Research Faculty Summit 2003

Outline

- 1st generation tools
- 2nd generation tools (sound, declarative)
  - SLAM
  - ESP
  - Fugue
- 3rd generation tools
- Conclusion

Microsoft Research Faculty Summit 2003

SPT Research Tools

- Sound & declaratively extensible
- SLAM
  - software model checking
- ESP
  - scalable error detection
- Vault/Fugue
  - behavioral types
- Being deployed in Microsoft

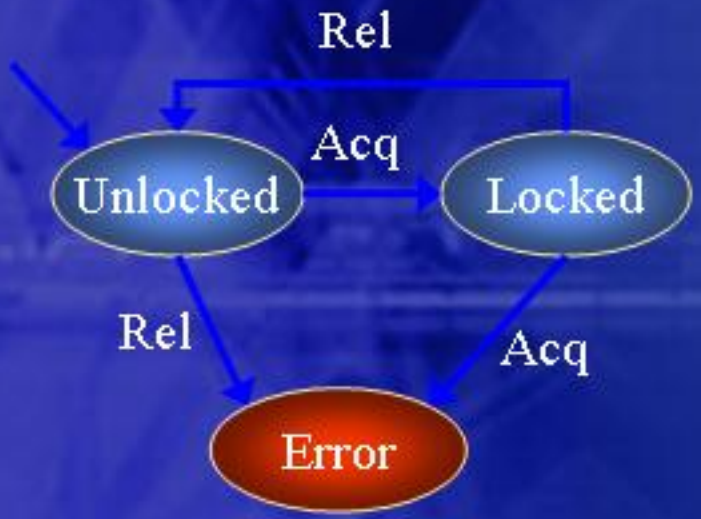
Microsoft Research Faculty Summit 2003

SLAM

- Software model checking
- Input
  - C source code "as is"
  - API rules in SLIC language
- Automatically create abstraction of C program
  - abstract model = Boolean program
- Systematic exploration of model's state space
  - does feasible path lead to error state in SLIC spec?
- Demand-driven refinement of model
  - exclude infeasible paths

Microsoft Research Faculty Summit 2003

State Machine for Locking

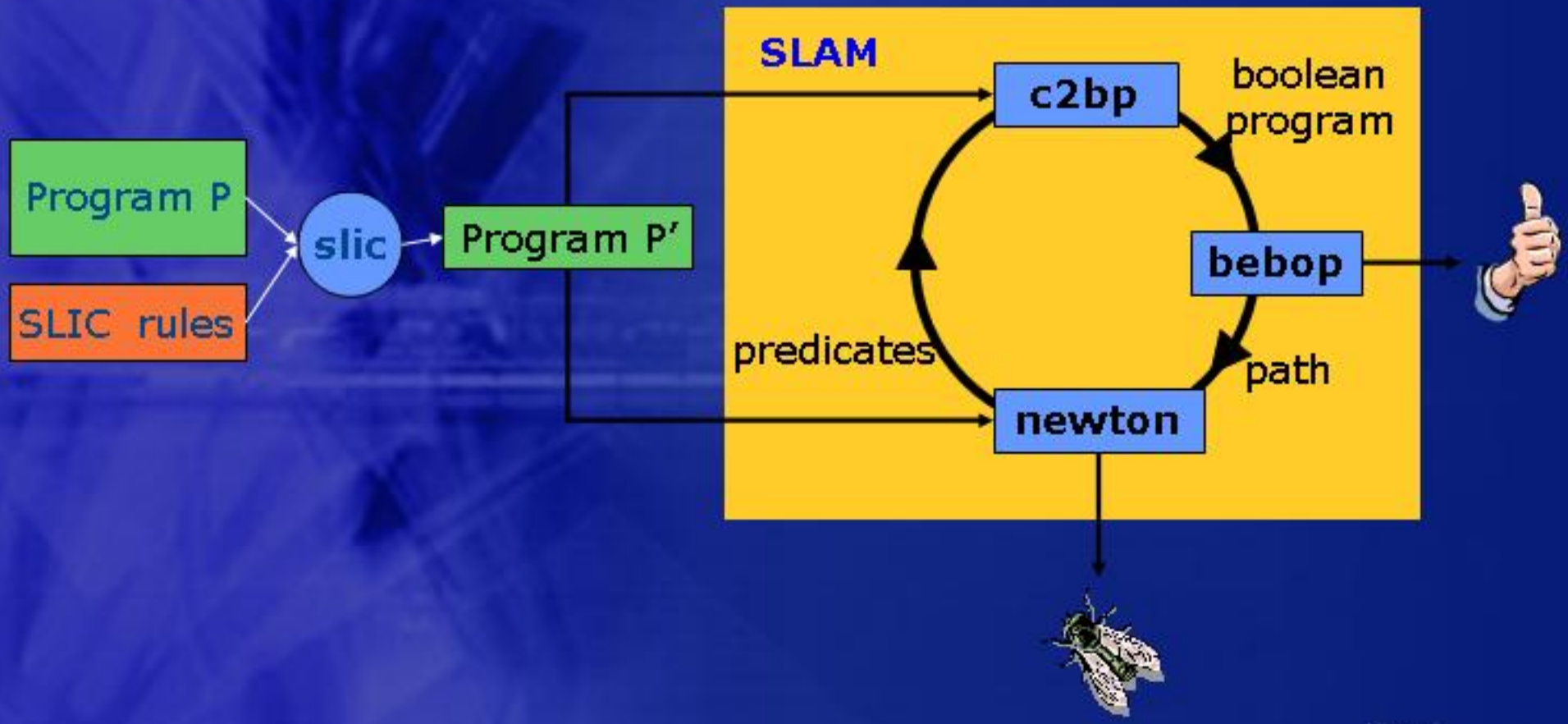


Locking Rule in SLIC

```
state {
  enum {Locked,Unlocked}
       s = Unlocked;
}
KeAcquireSpinLock.entry {
  if (s==Locked) abort;
  else s = Locked;
}
KeReleaseSpinLock.entry {
  if (s==Unlocked) abort;
  else s = Unlocked;
}
```

Microsoft Research Faculty Summit 2003

The SLAM Process



Microsoft Research Faculty Summit 2003

Example

```
do {
   KeAcquireSpinLock();

   nPacketsOld = nPackets;

   if(request){
       request = request->Next;
       KeReleaseSpinLock();
       nPackets++;
   }
} while (nPackets != nPacketsOld);

KeReleaseSpinLock();
```

Does this code obey the locking rule?

Microsoft Research Faculty Summit 2003

Example




```
do {
   KeAcquireSpinLock();



   if(*){

       KeReleaseSpinLock();

   }
} while (*);

KeReleaseSpinLock();
```

Model checking boolean program (bebop)

Microsoft Research Faculty Summit 2003

Example




```
do {
   KeAcquireSpinLock();

   nPacketsOld = nPackets;

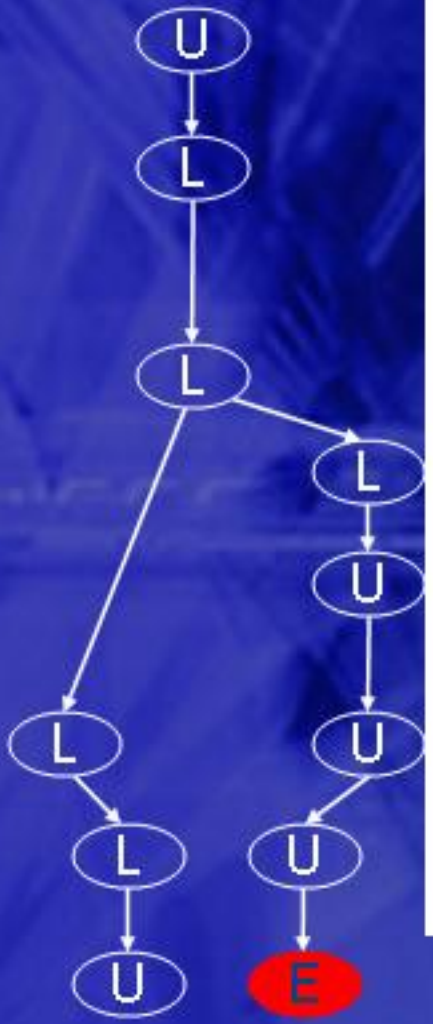
   if(request){
       request = request->Next;
       KeReleaseSpinLock();
       nPackets++;
   }
} while (nPackets != nPacketsOld);

KeReleaseSpinLock();
```

Is error path feasible in C program?

Microsoft Research Faculty Summit 2003

Example




```
do {
   KeAcquireSpinL
   nPacketsOld = nPackets; b := true;

   if(request){
       request = request->Next;
       KeReleaseSpinLock();
       nPackets++; b := b? false : *;
   }
} while (nPackets != nPacketsOld); !b

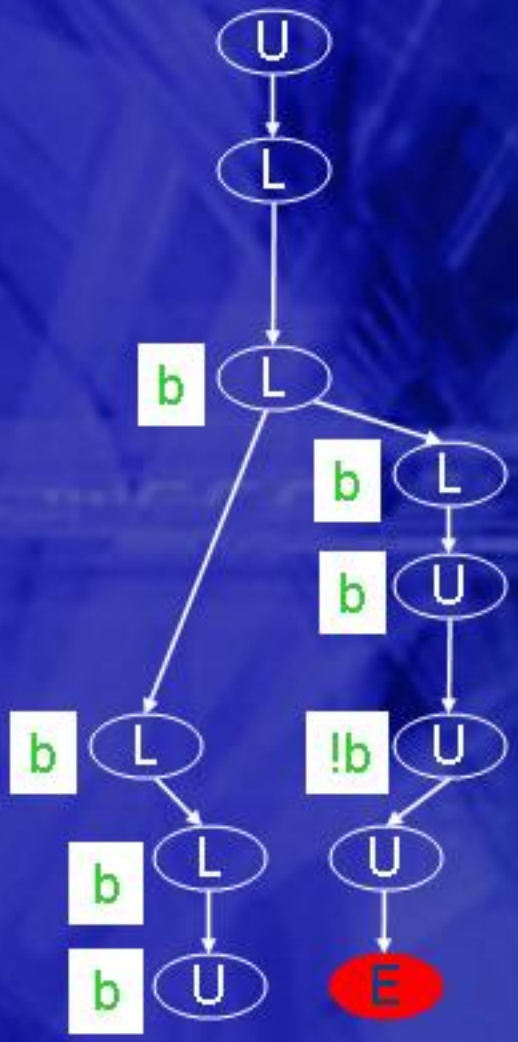
KeReleaseSpinLock();
```

Add new predicate
to boolean program

b : (nPacketsOld == nPackets)

Microsoft Research Faculty Summit 2003

Example




```
do {
   KeAcquireSpinLock();

   b := true;

   if(*){

      KeReleaseSpinLock();
      b := b? false : *;
   }
} while ( !b );

KeReleaseSpinLock();
```

Model checking refined boolean program

Microsoft Research Faculty Summit 2003

Example




```
do {
   KeAcquireSpinLock();

   b := true;

   if(*){

       KeReleaseSpinLock();
       b := b? false : *;
   }
} while ( !b );

KeReleaseSpinLock();
```

Model checking refined boolean program

Microsoft Research Faculty Summit 2003

SLAM Summary

- Software model checking
- Automatic analysis (no code annotation)
  - SLIC specification of API rules
- Report errors with source-level paths
- Very precise analysis to minimize false positives
- Applying to device drivers (SDV)
  - 100+ drivers, 13 rules, 60+ bugs
  - co-development with Windows Driver Quality Group

Microsoft Research Faculty Summit 2003

ESP

- Systematically find errors in large programs (10+ MLOC)
  - scalable and accurate program analysis
  - trade little precision for much scalability
  - sacrifice specification complexity
- ESP:
  - scalable whole-system value flow analysis
  - scalable module-level path-sensitive analysis
  - finite-state property specification

Microsoft Research Faculty Summit 2003

Analysis Dilemma

- Verification demands precise analysis
  - analyze each execution path to minimize false reports
- Big programs demand scalability
  - exponentially/infinitely many paths
  - cannot follow each execution path
  - must use approximate analysis

Microsoft Research Faculty Summit 2003

Path Sensitive Analysis

- Rules expressed as finite-state automata (FSA)
- Tool symbolically evaluates program
  - along each path, track
    - FSA state
    - program state
  - watch for FSA transitions into error state

Microsoft Research Faculty Summit 2003

Tracking Across Branches

- Does program state determines branch direction?
  - yes: process appropriate branch
  - no: process both branches, updating state
- Exponential growth in state at branches

Microsoft Research Faculty Summit 2003

Remedy 1: Dataflow Analysis

- Track only FSA state
- Ignore code that does not affect FSA
- At control flow joins, merge FSA states

Microsoft Research Faculty Summit 2003

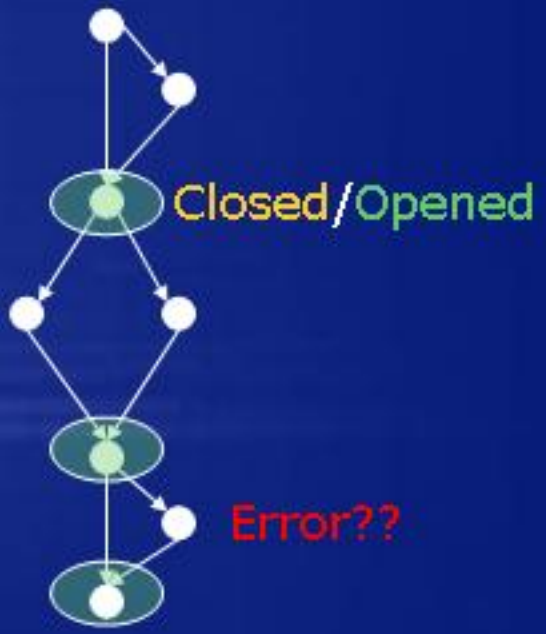
Too Imprecise




```
void main ()
{
 if (dump)
   Open();

 if (p)
   x = 0;
 else
   x = 1;

 if (dump)
   Close();
}
```



Microsoft Research Faculty Summit 2003

When Must Program State Be Tracked?

- Precise answer
  - when branch determines rule state
- Heuristic answer
  - when FSA moves into different states along two arms of branch

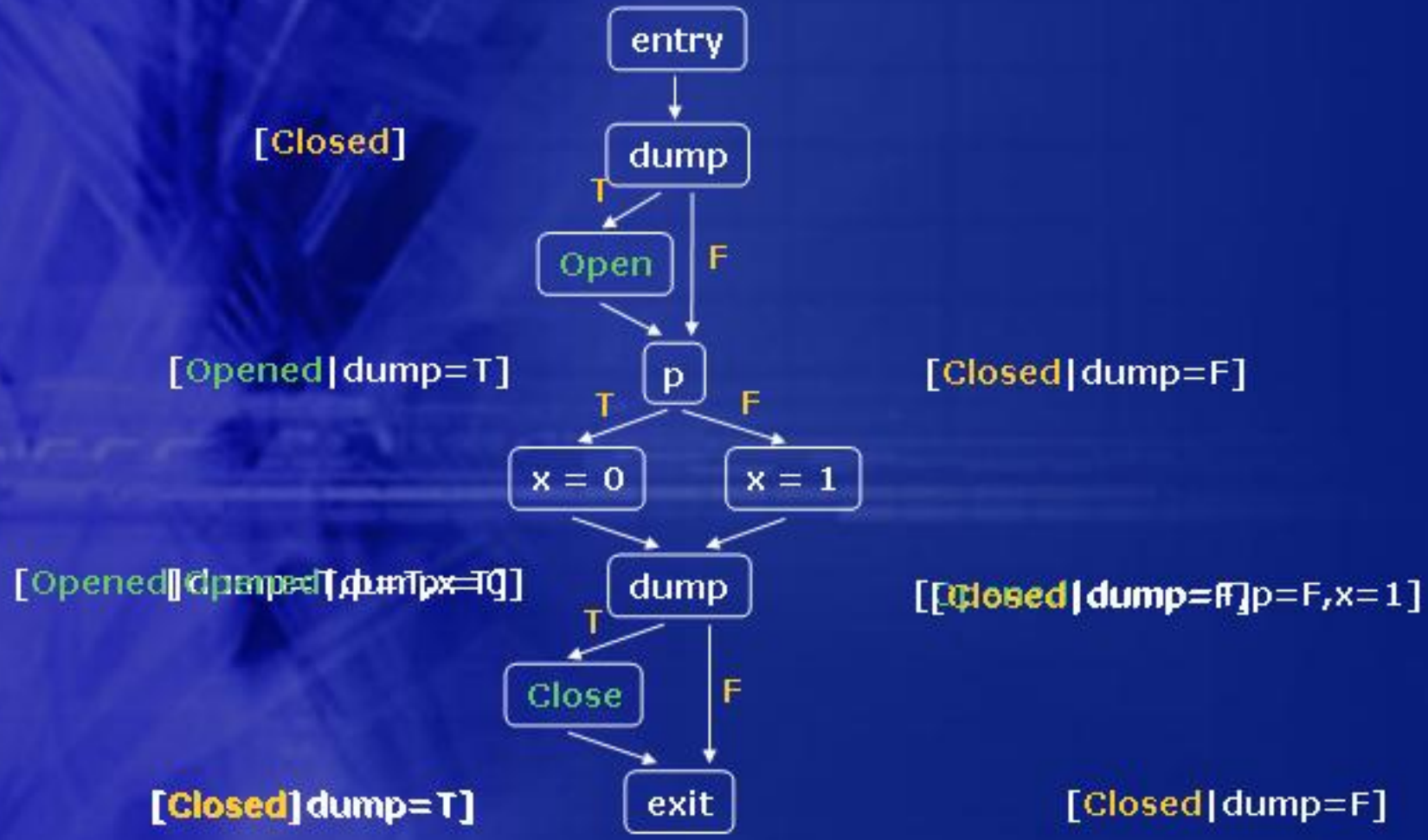
Microsoft Research Faculty Summit 2003

ESP Property Simulation

- Efficient (heuristic) path-sensitive analysis
- At control flow join points:
  - agreement on property FSA state?
    - yes: merge program states
    - no: retain both program states
- In practice, little code affects particular object's FSA state
  - exponential growth not a problem

Microsoft Research Faculty Summit 2003

Example



Microsoft Research Faculty Summit 2003

Fugue

- Make common API specifications statically checkable
  - can this method return null?
  - who owns this resource? do I have to free it?
  - do I have to call these methods in a particular order?
  - do these fields have data invariants I have to obey?
- Make documented rules part of the API itself

```
/// <param name="url">the address of a web page to fetch</param>
/// <returns><para>A string containing the content of
/// the web page at the <paramref name="url"/></para></returns>
/// <exception cref="System.ArgumentNullException">
///  <paramref name="url"/> is <see langword="null"/>.
/// </exception>
public string GetPage (string url);
```

Research Faculty Summit 2003

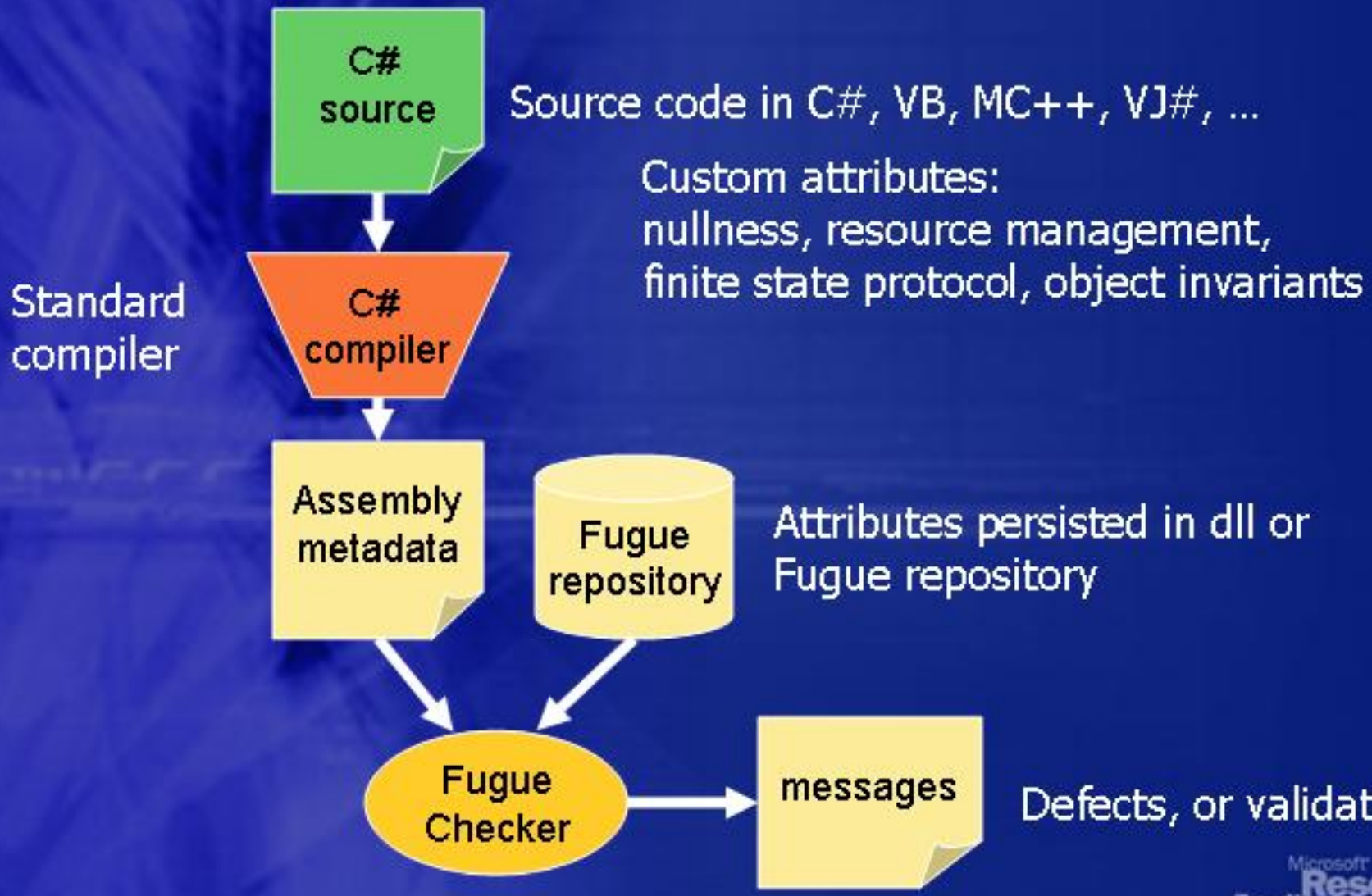
Fugue

- Make common API specifications statically checkable
  - can this method return null?
  - who owns this resource? Do I have to free it?
  - do I have to call these methods in a particular order?
  - do these fields have data invariants I have to obey?
- Make documented rules part of the API itself

```
[return:NotNull]
public string GetPage ([NotNull] string url);
```

Research Faculty Summit 2003

Overview



Microsoft Research Faculty Summit 2003

Enforces Contracts

- Attributes express contract

```
[return:NotNull] string GetPage ([NotNull] string url);
```

- Client checked against contract

```
static void Main (string[] args)
{
  WebCache cache = new WebCache();
  string homePage = Environment.GetEnvironmentVariable("HOMEPAGE");
  cache.GetPage(homePage);
  ....
```

parameter 1 to WebCache.GetPage is marked NotNull,
but actual 'homePage' may be null

Research Faculty Summit 2003

Enforces Contracts

- Attributes express contract

```
[return:NotNull] string GetPage ([NotNull] string url);
```

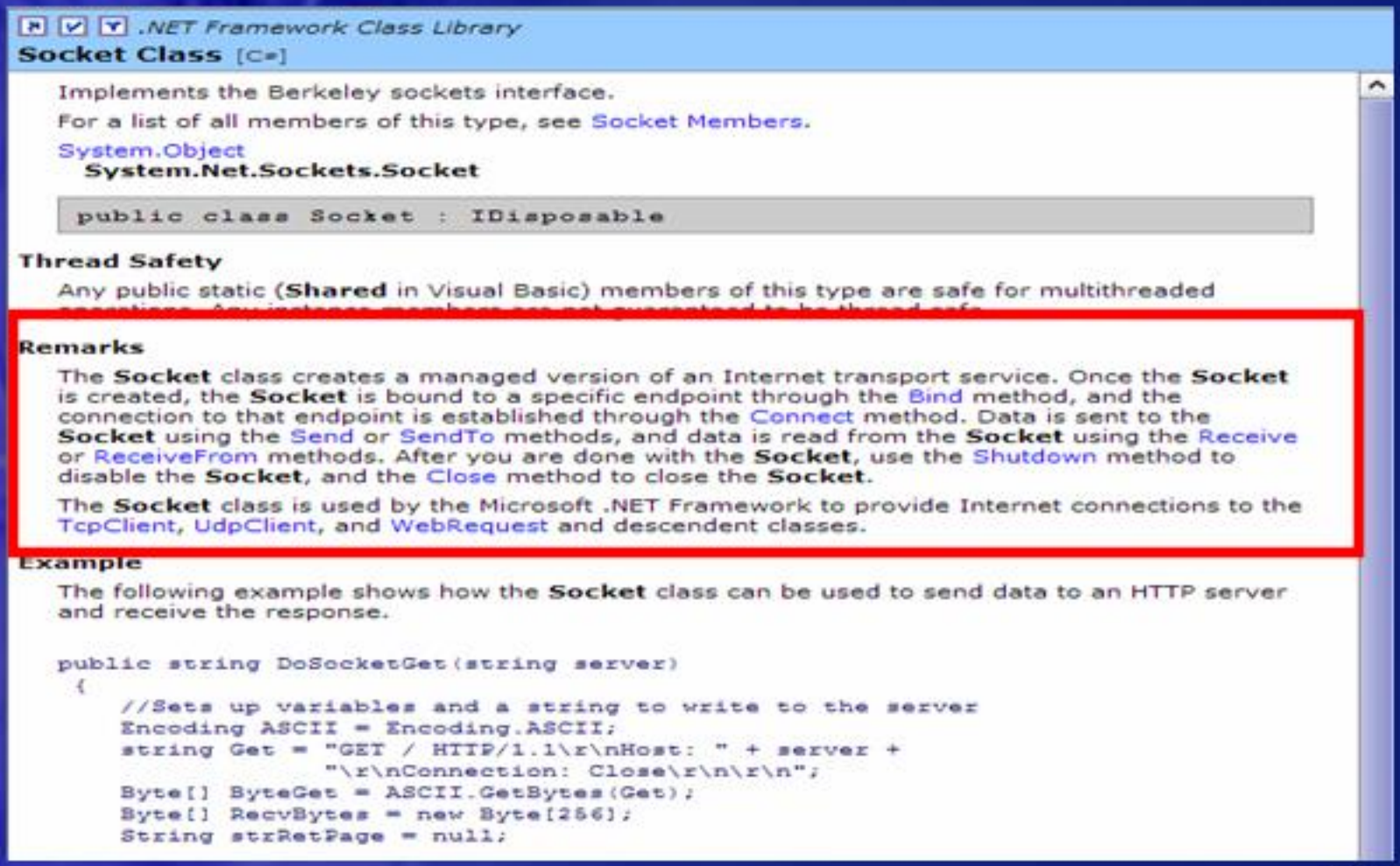
- Implementation checked against contract

```
[return:NotNull] string GetPage ([NotNull] string url)
{
  ...
  return cachedPage;
}
```

return value may be null,
but expected NotNull

Microsoft Research Faculty Summit 2003

API Usage Rules



.NET Framework Class Library

Socket Class [C#]

Implements the Berkeley sockets interface.

For a list of all members of this type, see Socket Members.

System.Object
  System.Net.Sockets.Socket

```
public class Socket : IDisposable
```

Thread Safety

Any public static (**Shared** in Visual Basic) members of this type are safe for multithreaded

Remarks

The **Socket** class creates a managed version of an Internet transport service. Once the **Socket** is created, the **Socket** is bound to a specific endpoint through the Bind method, and the connection to that endpoint is established through the Connect method. Data is sent to the **Socket** using the Send or SendTo methods, and data is read from the **Socket** using the Receive or ReceiveFrom methods. After you are done with the **Socket**, use the Shutdown method to disable the **Socket**, and the Close method to close the **Socket**.

The **Socket** class is used by the Microsoft .NET Framework to provide Internet connections to the TcpClient, UdpClient, and WebRequest and descendent classes.

Example

The following example shows how the **Socket** class can be used to send data to an HTTP server and receive the response.

```
public string DoSocketGet(string server)
 {
    //Sets up variables and a string to write to the server
    Encoding ASCII = Encoding.ASCII;
    string Get = "GET / HTTP/1.1\r\nHost: " + server +
                 "\r\nConnection: Close\r\n\r\n";
    Byte[] ByteGet = ASCII.GetBytes(Get);
    Byte[] RecvBytes = new Byte[256];
    String strRetPage = null;
```

Microsoft Research Faculty Summit 2003

API Usage Rules

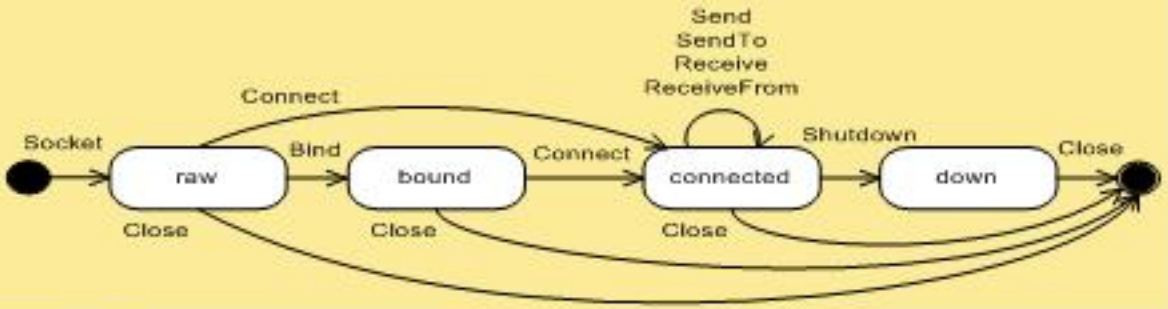
.NET Framework Class Library

Socket Class [C#]

Implements the Berkeley sockets interface.

For a list of all members of this type, see Socket Members.

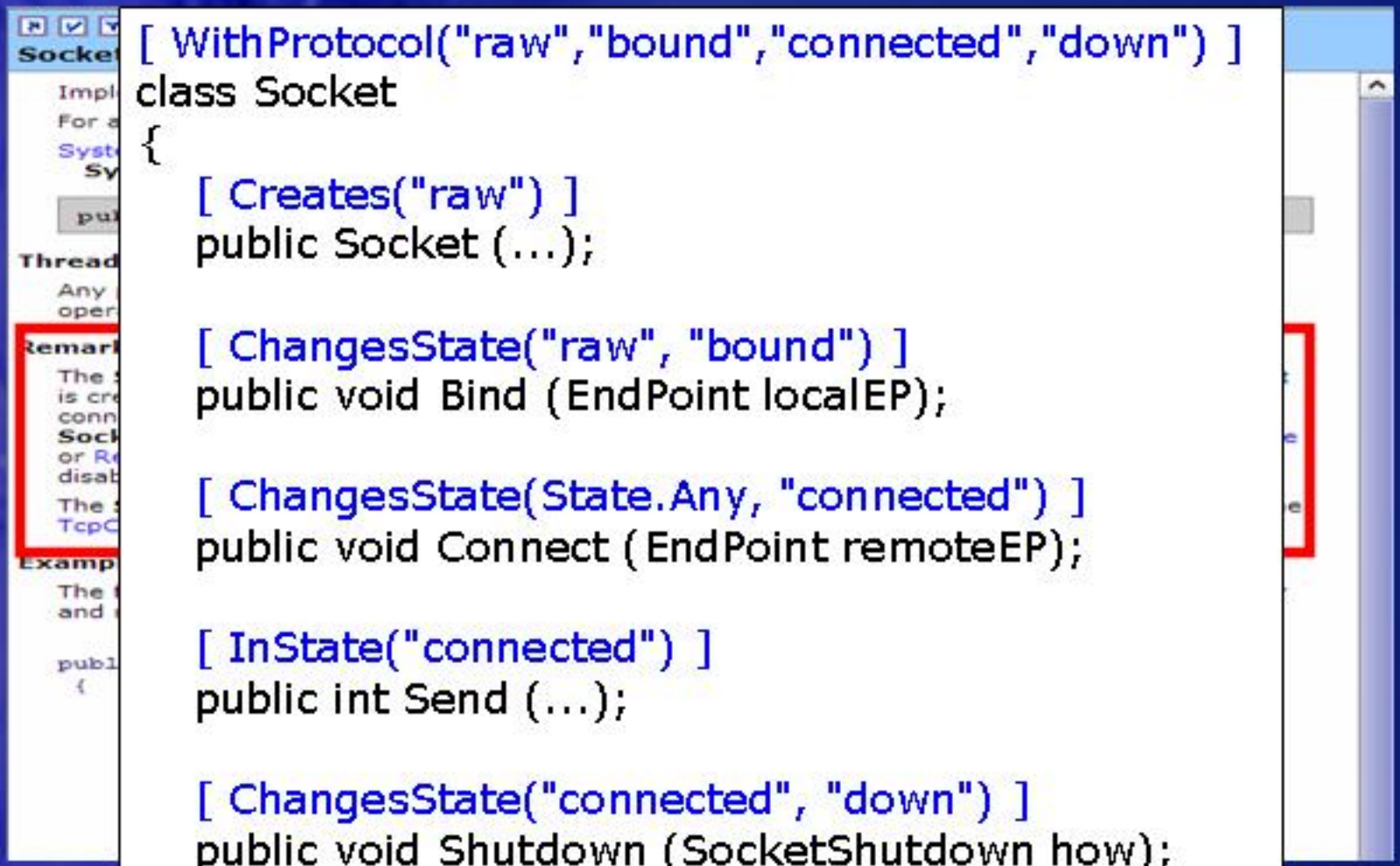
System.Object
  System.Net.Sockets.Socket




```
public string DoSocketGet(string server)
 {
     //Sets up variables and a string to write to the server
     Encoding ASCII = Encoding.ASCII;
     string Get = "GET / HTTP/1.1\r\nHost: " + server +
                  "\r\nConnection: Close\r\n\r\n";
     Byte[] ByteGet = ASCII.GetBytes(Get);
     Byte[] RecvBytes = new Byte[256];
     String strRetPage = null;
```

Microsoft Research Faculty Summit 2003

API Usage Rules



```
[ WithProtocol("raw","bound","connected","down") ]
class Socket
{
    [ Creates("raw") ]
    public Socket (...);

    [ ChangesState("raw", "bound") ]
    public void Bind (EndPoint localEP);

    [ ChangesState(State.Any, "connected") ]
    public void Connect (EndPoint remoteEP);

    [ InState("connected") ]
    public int Send (...);

    [ ChangesState("connected", "down") ]
    public void Shutdown (SocketShutdown how);
}
```

Microsoft Research Faculty Summit 2003

Extended Type Checking

- Fugue works like a type checker
  - checks each class, one method at a time
  - checks call using callee's signature/attributes
  - checks all paths, including some infeasible ones
  - fast enough to run after every compile

Microsoft Research Faculty Summit 2003

Continual Checking

- Provides contracts beyond types
  - nullness, resource management, abstract state protocols, class invariants
- Ensures consistency of code and contract
- Run as part of every compile
- Clarify rules for using APIs
  - standard, machine-readable documentation
  - documentation is checked
- Clear up responsibility for bugs
  - assigns blame based on contract

Microsoft Research Faculty Summit 2003

Outline

- 1st generation tools
- 2nd generation tools
- 3rd generation tools (targeted)
  - security
  - concurrency
- Conclusion

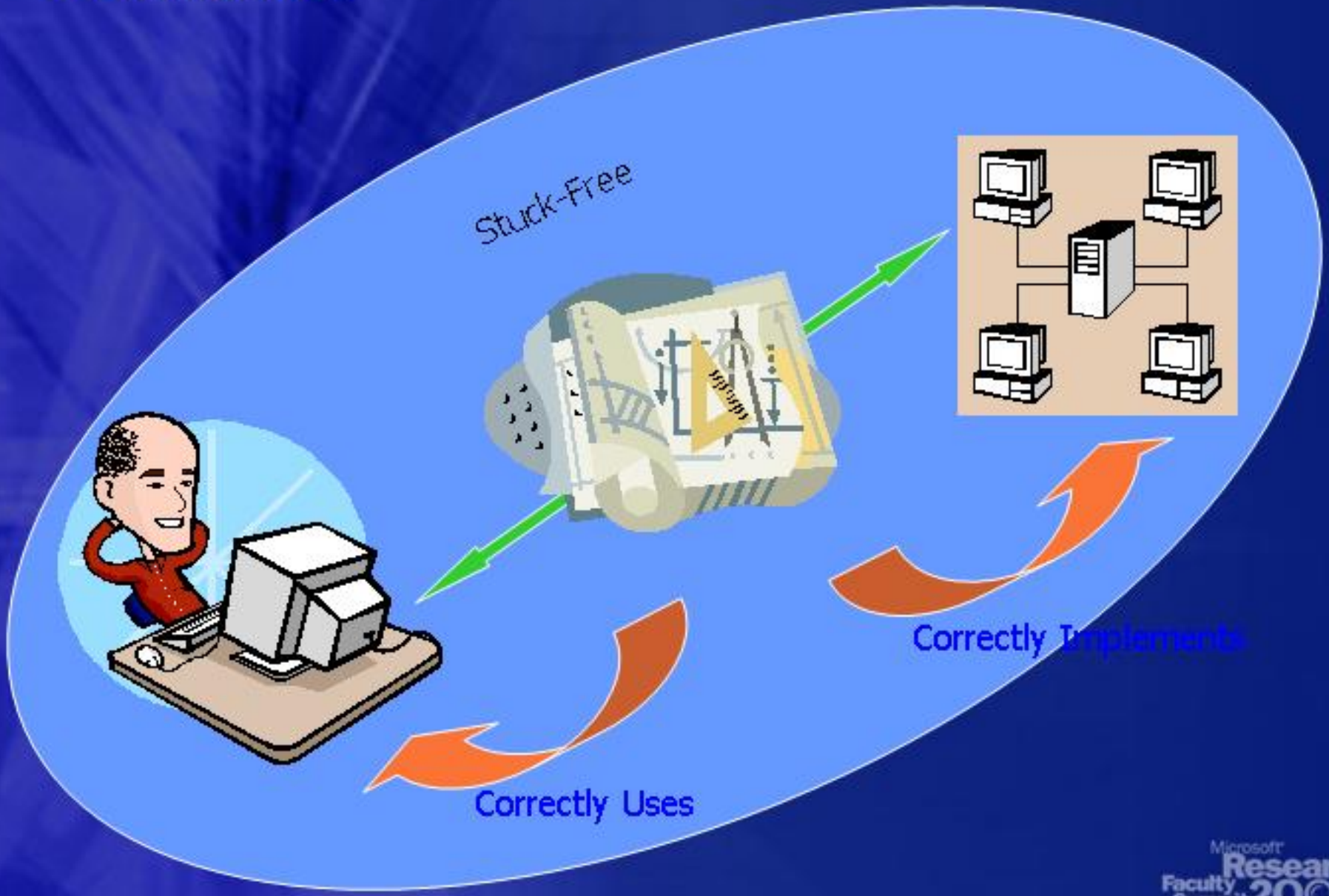
Microsoft Research Faculty Summit 2003

Directions for New Tools

- Specialized problems
  - security
  - concurrency
- Combine static analysis and run-time
- New analysis techniques
  - theorem proving
  - SAT solvers

Microsoft Research Faculty Summit 2003

Behave!



Microsoft Research Faculty Summit 2003

Outline

- 1st generation tools
- 2nd generation tools
- 3rd generation tools
- Conclusion

Microsoft Research Faculty Summit 2003

Lessons From Tools

- Heuristics suffice
  - users happy to find (some) bugs
  - soundness is additional benefit
- User interface is crucial
  - PREfix spent more effort filtering than finding bugs
- Developers are rational
  - usage based on cost-benefit tradeoff
  - do I expect this tool to save time & effort?

Microsoft Research Faculty Summit 2003

Tool Adoption

- Simple tools pave way for more sophisticated tools
  - change developer attitude
  - eliminating simple bugs exposes more complicated ones

Microsoft Research Faculty Summit 2003

Conclusion

- Tools improve program development
  - find significant bugs in huge code bases
  - potential to eliminate classes of bugs
- Start of a new way of programming
  - crude and inaccurate tools find limited class of bugs
  - but, already improve developer & tester productivity
- Starting point for better software

Microsoft Research Faculty Summit 2003

http://research.microsoft.com/spt
http://research.microsoft.com/pprc

Microsoft Research Faculty Summit 2003



© 2002 Microsoft Corporation. All rights reserved.
This presentation is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY.